U.S. PATENT APPLICATION FOR

SYSTEM AND METHOD FOR CONFLICT RESPONSES IN A CACHE COHERENCY PROTOCOL

Inventor(s):

Stephen R. Van Doren
8 Iroquois Drive
Northborough, MA 01532

Gregory E. Tierney
161 Boston Road
Chelmsford, MA 01824

Simon C. Steely, Jr.
8 Anna Louise Drive
Hudson, NH 03051

Attorney Docket No. 200313631-1

SYSTEM AND METHOD FOR CONFLICT RESPONSES IN A CACHE COHERENCY PROTOCOL

RELATED APPLICATIONS

[0001]   This application is related to the following commonly assigned co-pending patent applications entitled:

[0002]   "CACHE COHERENCY PROTOCOL WITH ORDERING POINTS," Attorney Docket No. 200313588-1; "SYSTEM AND METHOD FOR RESOLVING TRANSACTIONS IN A CACHE COHERENCY PROTOCOL," Attorney Docket No. 200313589-1; "SYSTEM AND METHOD TO FACILITATE ORDERING POINT MIGRATION," Attorney Docket No. 200313612-1; "SYSTEM AND METHOD TO FACILITATE ORDERING POINT MIGRATION TO MEMORY," Attorney Docket No. 200313613-1; "SYSTEM AND METHOD FOR CREATING ORDERING POINTS," Attorney Docket No. 200313614-1; "SYSTEM AND METHOD FOR CONFLICT RESPONSES IN A CACHE COHERENCY PROTOCOL WITH ORDERING POINT MIGRATION," Attorney Docket No. 200313615-1; "SYSTEM AND METHOD FOR READ MIGRATORY OPTIMIZATION IN A CACHE COHERENCY PROTOCOL," Attorney Docket No. 200313616-1; "SYSTEM AND METHOD FOR BLOCKING DATA RESPONSES," Attorney Docket No. 200313628-1; "SYSTEM AND METHOD FOR NON-MIGRATORY REQUESTS IN A CACHE COHERENCY PROTOCOL," Attorney Docket No. 200313629-1; "SYSTEM AND METHOD FOR CONFLICT RESPONSES IN A CACHE COHERENCY PROTOCOL WITH ORDERING POINT MIGRATION," Attorney Docket No. 200313630-1; "SYSTEM AND METHOD FOR RESPONSES BETWEEN DIFFERENT CACHE COHERENCY PROTOCOLS," Attorney Docket No. 200313632-1, all of which are filed contemporaneously herewith and are incorporated herein by reference.

BACKGROUND

[0003]   Multiprocessor systems employ two or more computer processors that can communicate with each other, such as over a bus or a general interconnect network. In such systems, each processor may have its own memory cache (or cache store) that is separate from the main system memory that the individual processors can access. Cache memory connected to each processor of the computer system can often enable fast access to data. Caches are useful because they tend to reduce latency associated with accessing

data on cache hits, and they work to reduce the number of requests to system memory. In particular, a write-back cache enables a processor to write changes to data in the cache without simultaneously updating the contents of memory. Modified data can be written back to memory at a later time.

[0004] Coherency protocols have been developed to ensure that whenever a processor reads a memory location, the processor receives the correct or true data. Additionally, coherency protocols help ensure that the system state remains deterministic by providing rules to enable only one processor to modify any part of the data at any one time. If proper coherency protocols are not implemented, however, inconsistent copies of data can be generated.

[0005] There are two main types of cache coherency protocols, namely, a directory-based coherency protocol and a broadcast-based coherency protocol. A directory-based coherency protocol associates tags with each memory line. The tags can contain state information that indicates the ownership or usage of the memory line. The state information provides a means to track how a memory line is shared. Examples of the usage information can be whether the memory line is cached exclusively in a particular processor's cache, whether the memory line is shared by a number of processors, or whether the memory line is currently cached by any processor.

[0006] A broadcast-based coherency protocol employs no tags. Instead, in a broadcast-based coherency protocol, each of the caches monitors (or snoops) requests to the system. The other caches respond by indicating whether a copy of the requested data is stored in the respective caches. Thus, correct ownership and usage of the data are determined by the collective responses to the snoops.

SUMMARY

[0007] One embodiment of the present invention may comprise a system that includes a first node that provides a broadcast request for data. The first node receives a read conflict response to the broadcast request from the first node. The read conflict response indicates that a second node has a pending broadcast read request for the data. A third node provides the requested data to the first node in response to the broadcast request from the first node. The first node fills the data provided by the third node in a cache associated with the first node.

[0008] Another embodiment of the present invention may comprise a multi-processor network that includes a first processor node operative to issue a first source

2

broadcast request for data. A second processor node is operative to issue a second source broadcast request for the data. A third node is operative to provide a data response in response to the respective source broadcast requests of the first and second processor nodes. The third node is one of an owner processor node and a memory node. The second processor node is operative to provide a read conflict response to the first source broadcast request when the second source broadcast request is a read request. The second processor node is operative to provide a second conflict response to the first source broadcast request when the second source broadcast request is a write request. The first processor node is operative in response to receiving a read conflict response from the second processor to implement a cache fill with the data provided by the third node.

[0009]   Another embodiment of the present invention may comprise a computer system that includes a first processor operative to issue a source broadcast request for data. A second processor is operative to issue a source broadcast request for the data. A node is operative to provide a data response to both the first and second processors in response to the source broadcast requests of the first and second processors. The second processor in response to the source broadcast request of the first processor provides a read conflict response when the source broadcast request of the second processor is a source broadcast read request. The second processor in response to the source broadcast request of the first processor provides a second conflict response when the source broadcast request of the second processor is a source broadcast write request. The first processor in response to the read conflict response of the second processor is operative to fill the data provided by the third node in a cache associated with the first processor.

[0010]   Yet another embodiment of the present invention may comprise a method that includes providing a source broadcast request from a first node for data. The method also includes providing a read conflict response to the first node from a second node in response to the source broadcast request from the first node, the read conflict response indicating that the second node has a pending broadcast read request for the data. The method also includes providing the requested data to the first node from a third node in response to the source broadcast request from the first node. The method further includes placing the data provided by the third node in a cache associated with the first node.

[0011]   Still another embodiment of the present invention may comprise a computer system that includes a hybrid cache coherency protocol that employs source broadcast protocol mode and a forward progress protocol mode. The computer system is operative to fill a cache line associated with a source node with requested data provided in

response to a source broadcast protocol mode request for the data when there is a source broadcast protocol read conflict with another node in the computer system. The computer system is further operative to reissue a request for the data from a source node using a forward progress protocol mode request for the data when there is a source broadcast protocol second conflict with another node in the computer system.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012]   FIG. 1 depicts an example of a multi-processor system.

[0013]   FIG. 2 depicts an example of a data state flow diagram that may be implemented in a coherency protocol.

[0014]   FIG. 3 depicts an example of a conflict state flow diagram that may be implemented in a coherency protocol.

[0015]   FIG. 4 depicts an example of another multi-processor system.

[0016]   FIG. 5 depicts an example of a processor within a multi-processor system.

[0017]   FIG. 6 depicts a first example conflict scenario illustrating state transitions for a coherency protocol.

[0018]   FIG. 7 depicts a second example conflict scenario illustrating state transitions for a coherency protocol.

[0019]   FIG. 8 depicts a third example conflict scenario illustrating state transitions for a coherency protocol.

[0020]   FIG. 9 depicts a fourth example conflict scenario illustrating state transitions for a coherency protocol.

[0021]   FIG. 10 depicts a fifth example conflict scenario illustrating state transitions for a coherency protocol.

[0022]   FIG. 11 depicts a flow diagram illustrating a method.

DETAILED DESCRIPTION

[0023] This disclosure relates generally to a hybrid cache coherency protocol, such as a broadcast source snoop protocol (SSP) implemented in conjunction with a forward progress (*e.g.*, directory-based or null-directory) protocol (FPP). Characteristic of the hybrid cache coherency protocol, requests for data are initially transmitted broadcast using SSP broadcast snoop requests. If the snoop requests fail or otherwise cannot be completed, such as where there is a conflict between multiple processors attempting to read and/or write the same cache line, the protocol can transition to the FPP mode and the requests can be reissued using FPP request commands. Other forward progress techniques could also be utilized.

[0024] The cache coherency protocol employs conflict states that are assigned to a miss address file (MAF) entry for an outstanding broadcast snoop request. The conflict states are used to determine how to handle conflicts that arise in broadcast snoop request transactions. The conflict states include a read conflict (RD-CONF) state and a conflict (CONFLICT) state. In general, the RD-CONF state is assigned to a MAF entry in a conflict scenario in which the broadcast snoop requests that conflict with the MAF entry are broadcast read snoop requests. In general, the CONFLICT state is assigned to a MAF entry in a conflict scenario in which the broadcast snoop requests that conflict with the MAF entry include broadcast write snoop requests.

[0025] The implementation of the CONFLICT and RD-CONF states is useful in multi-processor systems employing a hybrid cache coherency protocol, such as the SSP/FPP hybrid cache coherency protocol described herein. In a conflict scenario in which a source processor receives a data response and a RD-CONF response to a broadcast snoop request for data, the source processor can place the data in a cache associated with the source processor. In a conflict scenario in which a source processor receives a data response and a CONFLICT response to a broadcast snoop request for data, the source processor can employ a forward progress technique to complete the transaction. For example, the source processor can transition to a forward progress protocol (FPP) mode and reissue the request for the data using FPP request commands. The cache coherency protocol disclosed herein thus mitigates having to transition to the FPP mode in certain conflict scenarios, which can help reduce latency.

[0026] FIG. 1 depicts an example of a system 10 in which a cache coherency protocol of the present invention may be implemented. The system 10 illustrates a multi-

5

processor environment that includes a plurality of processors 12 and 14 (indicated at PROCESSOR 1 through PROCESSOR N, where N is a positive integer (N>1)). The system 10 also includes memory 16 that provides a shared address space. For example, the memory 16 can include one or more memory storage devices (*e.g.*, dynamic random access memory (DRAM)).

[0027]   The processors 12 and 14 and memory 16 define nodes in the system that can communicate with each other *via* requests and corresponding responses through a system interconnect 18. For example, the system interconnect 18 can be implemented as a switch fabric or a hierarchical switch. Also associated with the system 10 can be one or more other nodes, indicated schematically at 20. The other nodes 20 can correspond to one or more other multi-processor systems connected to the system interconnect 18, such as through an appropriate interconnect interface (not shown).

[0028]   Each of the processors 12 and 14 includes at least one corresponding cache 22 and 24. For purposes of brevity, each of the respective caches 22 and 24 is depicted as unitary memory devices, although the caches may include a plurality of memory devices or different cache levels. Each of the caches 22 and 24 includes a plurality of cache lines. Each cache line has an associated tag address that identifies corresponding data stored in the line. The cache lines can also include information identifying the state of the data for the respective lines.

[0029]   The system 10 thus employs the caches 22 and 24 and the memory 16 to store blocks of data, referred to herein as "memory blocks." A memory block can occupy part of a memory line, an entire memory line or span across multiple lines. For purposes of simplicity of explanation, however, it will be assumed that a "memory block" occupies a single "memory line" in memory or a "cache line" in a cache. Additionally, a given memory block can be stored in a cache line of one or more caches as well as in a memory line of the memory 16.

[0030]   Each cache line can also include information identifying the state of the data stored in the respective cache. A given memory block can be stored in a cache line of one or more of the caches 22 and 24 as well as in a memory line of the memory 16, depending on the state of the line. Whether a cache line contains a coherent copy of the data also depends on the state of the cache line. Certain states employed by the coherency protocol can define a given cache line as an ordering point for the system 10 employing a broadcast-based protocol. An ordering point characterizes a serialization of requests to the same memory line (or memory block) that is understood and followed by the system 10.

[0031] The system 10 implements the cache coherency protocol described herein to manage the sharing of memory blocks so as to help ensure coherence of data. The cache coherency protocol of the system 10 utilizes a plurality of states to identify the state of each memory block stored in respective cache lines of the caches 22 and 24 and the memory 16. The coherency protocol establishes rules for transitioning between states, such as if data is read from or written to memory 16 or one of the caches 22 and 24.

[0032] As used herein, a node that issues a request, such as a read or write request, defines a source node. Other nodes within the system 10 are potential targets of the request. Additionally, each memory block in the system 10 can be assigned a home node that maintains necessary global information and a data value for that memory block. When a source node issues a source broadcast snoop request for data, an entry associated with the request is allocated in a miss address file (MAF). The MAF maintains information associated with, for example, the tag address of the data being requested, the type of request, and response information received from other nodes in response to the request. The MAF entry for the request is maintained until the request associated with the MAF is complete.

[0033] For example, when a source node, such as the processor 12, requires a copy of a given memory block, the source node typically first requests the memory block from its local, private cache by identifying the tag address associated with the memory block. If the data is found locally, the memory access is resolved without communication *via* the system interconnect 18. When the requested memory block is not found locally, the source node 12 can request the memory block from the system 10, including the memory 16. In addition to the request identifying a tag address associated with the requested memory block, the request usually identifies the type of request or command being issued by the requester. Whether the other nodes 14 and the memory 16 will return a response depends upon the type of request, as well as the state of the identified memory block contained in the responding nodes. The cache coherency protocol implemented by the system 10 defines the available states and possible state transitions.

[0034] A set of cache states that can be included in the cache coherency protocol described herein is depicted below in Table 1. Each cache line of the respective caches 22 and 24 of the processors 12 and 14 may be associated or tagged with one of the cache states in table 1. Since there are eight possible states, the state information can be encoded by a three-bit data word, for example.

TABLE 1

STATE	DESCRIPTION
I	Invalid - The cache line does not exist.
S	Shared - The cache line is valid and unmodified by caching processor. Other processors may have valid copies, and the caching processor cannot respond to snoops by returning data.
E	Exclusive - The cache line is valid and unmodified by caching processor. The caching processor has the only cached copy in the system and may respond to snoops by returning data.
F	First (among equals) - The cache line is valid and unmodified by caching processor. Other processors may have valid copies, and caching processor may respond to snoops by returning data.
D	Dirty - The cache line is valid and more up-to-date than memory. The cache line has not been modified by the caching processor, and the caching processor has the only cached copy in the system. The caching processor must respond to snoops by returning data and must write data back to memory upon displacement. The dirty state permits a modified block to be transferred between caches without updating memory.
M	Modified - The cache line is valid and has been modified by the caching processor. The caching processor has the only cached copy in the system, and the caching processor must respond to snoops by returning data and must write data back to memory upon displacement.
O	Owned - The cache line is valid and more up-to-date than memory. The caching processor cannot modify the cache line. Other processors may have valid copies, and the caching processor must respond to snoops by returning data and must write data back to memory upon displacement.
T	Transition - The cache line is in transition. The cache line may be transitioning from O, M, E, F or D to I, or the line may be transitioning from I to any one of the valid states.

[0035]   As mentioned above, the state of a cache line can be utilized to define a cache ordering point in the system 10. In particular, for a protocol implementing the states set forth in Table 1, a processor including a cache line having one of the states M, O, E, F or D may be referred to as an owner processor or node. The owner node can serve as a cache ordering point for the data contained in the cache line for transactions in the broadcast-based protocol. An owner processor (*e.g.*, processor 12 or 14) that serves as the cache ordering point is capable of responding with data to snoops for the data. For example, processor 14 may be an owner processor for particular data and thus can provide a copy of the data to another cache 12. The type of data returned by an owner processor depends on the state of the data stored in the processor's cache. The response may also

vary based on the type of request as well as whether a conflict exists. The memory 16 seeks to return a copy of the data stored in the memory. The memory copy of the data is not always a coherent copy and may be stale (*e.g.*, when there is a modified copy of the data cached by another processor).

[0036] The cache coherency protocol described herein can provide for ordering point migration in which a cache ordering point is transferred from a target node to a source processor in response to a source broadcast read request. For example, a target node (*e.g.*, processor 14) including an M-state cache line can, in response to a source broadcast read request, provide an ownership data response to a source node (*e.g.*, processor 12), and the source node cache line transitions to the D-state. To mitigate the vulnerability of the ordering point during migration, the cache line of the target processor 14 can transition to the T-state while the ordering point migration is pending. Upon completion of the ordering point transfer, the target processor 14 cache line can transition from the T-state to the I-state. The ordering point is thus transferred (*i.e.*, the ordering point migrates) from the target processor 14 to the source processor 12.

[0037] Additionally, the source processor 12 can provide a message that acknowledges when the ordering point has successfully migrated (*e.g.*, a migration acknowledgement or "MACK" message). The cache line of the target processor 14 can further transition from the T-state to the I-state in response to receiving the MACK message from the source processor 12. The target processor 14 can respond to the MACK message by providing a further acknowledgement message back to the source processor 12 (*e.g.*, a MACK acknowledgement or MACK-ACK message). The source broadcast read request by the source processor 12 that initiated the migration sequence can be considered completed in response to receiving the MACK-ACK message from the target processor 14.

[0038] The processors 12 and 14 of the system 10 can obtain copies of desired data by issuing data requests in either the SSP or FPP portion of the cache coherency protocol implemented in the system. A list of example data requests that can be included in the SSP portion of the cache coherency protocol described herein, and thus issued through a source broadcast request by a processor (*e.g.*, processors 12 and 14), is depicted below in Table 2.

TABLE 2

Request Type	Request	Request Description
Reads	XREADN	Broadcast read line code: Non-migratory read request.
	XREADM	Broadcast read line data: Migratory read request.
	XREADC	Broadcast read current (non-coherent read).
Writes	XRDINVAL	Broadcast read and invalidate line with owner.
	XUPGRADE	Broadcast invalidate line - upgrade un-writable copy.
Memory Updates	XWRITE	Broadcast memory write-back - victim write.
	XUPDATE	Broadcast memory update - victim write.
	XWRITEC	Broadcast write coherent.
Special Commands	MACK	Broadcast migration acknowledgment.
	XINVAL	Broadcast invalidate.

[0039] According to the cache coherency protocol described herein, source processors 12 and 14 issue data requests initially as broadcast snoop requests using the SSP commands set forth in Table 2. If the snoop requests fail and a transition to the FPP is required (*e.g.*, due to a conflict), the system 10 can transition to FPP mode and the requests can be reissued using FPP commands.

[0040] Whenever a broadcast read or write snoop is issued by a source node (*e.g.*, source processor 12) in the system 10, target nodes of the system (*e.g.*, target processor 14, memory 16, and nodes 20) may issue an SSP response to the snoop. A list of example SSP responses that may be included in the cache coherency protocol described herein is depicted below in Table 3.

10

TABLE 3

SSP Broadcast Response (SSP)	Response Description
D-DATA	Ownership data response - Corresponding snoop command was the first to arrive at a cached ordering point (M, O, D, E, F state); the ordering point is being transferred to the requesting processor. At most, one D-DATA command can exist per cache line at any given time.
S-DATA	Shared data response - Data is being returned from a cached ordering point; the ordering point is not being transferred.
M-DATA	Memory data response - Data is being returned from home memory.
MISS	General snoop response: - Snoop failed to match a cache or MAF entry at a snoop target. - Snoop matched at a snoop target and invalidated a cache line at the target. - Acknowledgement for broadcast invalidate line requests. - Acknowledgement for broadcast migration acknowledgement requests. - Acknowledgement for broadcast victim write requests.
SHARED	Snoop hit shared - Read snoop matched on a cache line in the S-state.
CONFLICT	Snoop conflict - Snoop matched a valid write MAF (read or write) or T-state cache line at a target processor.
RD-CONF	Snoop read conflict - A special case conflict where a snoop matched a valid read MAF.
FPP	Snoop hit FPP-Mode MAF - Some other processor is trying to access the same cache line and has already transitioned to the forward progress protocol (FPP) mode. This response is required for forward progress/starvation avoidance.

[0041]   A target node can provide an ownership data response that includes D-DATA, for example, when the processor has an ownership state (*e.g.*, M, O, E, F or D) associated with the cached data in the SSP protocol. It is the state of the cached data that defines the node (processor) as a cache ordering point for the data. When a processor responds with D-DATA, the ordering point is transferred to the requesting processor. S-DATA is a shared data response that indicates data is being returned from a cached ordering point, although the ordering point itself is not being transferred to the requester. An S-DATA response also indicates that a copy of the data may be in one or more other caches. An M-DATA response can be provided by memory (*e.g.*, a home node) by returning the present value for the data stored in memory. It is possible that the M-DATA is stale and not up-to-date.

11

[0042] When a source node (*e.g.*, source processor 12) issues a source broadcast request for data, each of the target nodes (*e.g.*, target processor 14, memory 16, and nodes 20) may provide a data response. In the cache coherency protocol described herein, there are three different types of data responses: shared data responses (S-DATA), dirty data responses (D-DATA), and memory data responses (M-DATA). It is thus possible that, in response to a source broadcast request for data, the source processor 12 can receive several different data responses. Accordingly, the source processor 12 requester can employ a data state machine associated with the MAF entry for the source broadcast request to manage filling data in the cache of the source processor. FIG. 2 depicts an example of a data state diagram that represents operation of a data state machine that can be utilized to manage data responses returned to a source node in the SSP protocol. The example data state diagram of FIG. 2 implements the data responses set forth in Table 3.

[0043] As shown in the data state diagram of FIG. 2, D-DATA overrides both M-DATA and S-DATA, meaning that D-DATA will result in a cache fill, overwriting M-DATA or S-DATA that is received prior to the D-DATA. Additionally, S-DATA will overwrite M-DATA, but not D-DATA. Thus, D-DATA has priority over M-DATA and S-DATA, and S-DATA has priority over M-DATA. M-DATA results in a cache fill only if no other types of data have been received. If a lower priority data is received at a requester, the requester can drop the subsequent, lower priority data. Also, as shown in FIG. 2, if multiple S-DATA responses are received, a SET-CONF condition exists and a CONFLICT message is provided to the conflict state machine associated with the MAF.

[0044] Examples of processor snoop responses to source broadcast snoop requests that can occur in the system 10 and the target node transitions that result therefrom are provided in Table 4. The state transitions set forth in Table 4 assume that no conflicts are encountered in response to the respective commands. Conflict conditions can affect state transitions, as described herein. As shown in Table 4, the response to the source node varies depending on the type of broadcast snoop request received at the target node and the cache state of the target node when the snoop request is received.

TABLE 4

Source Node Request Type	Source Node Request	Target Node Cache State	Target Node Next Cache State	Response to Source Node
Reads	XREADN	T	Unchanged	Conflict
	XREADN	I	Unchanged	MISS
	XREADN	S	Unchanged	Shared
	XREADN	E, F	F	S-DATA
	XREADN	M, D, O	O	S-DATA
	XREADM	T	Unchanged	Conflict
	XREADM	I	Unchanged	MISS
	XREADM	S	Unchanged	Shared
	XREADM	E, F	F	S-DATA
	XREADM	D, O	O	S-DATA
	XREADM	M	T	D-DATA
	XREADC	T	Unchanged	Conflict
	XREADC	S, I	Unchanged	MISS
	XREADC	M, D, O, E, F	Unchanged	S-DATA
Writes	XRDINVAL	T	Unchanged	Conflict
	XRDINVAL	S, I	I	MISS
	XRDINVAL	M, D, O, E, F	T	D-DATA
	XUPGRADE	S, I	I	MISS
	XUPGRADE	M, D, O, E, F, T	Error - - XUPGRADE should not find an owner or T-state target node.	
Memory Updates	XWRITE	S, I	Unchanged	MISS
	XWRITE	M, D, O, E, F, T	Error - - XWRITE should not find an owner or T-state target node.	
Special Commands	MACK	T	I	MISS
	MACK	M, D, O, E, F, S, I	Error - - MACK should always find a T-state target node.	
	XINVAL	T, I	Unchanged	MISS
	XINVAL	M, D, O, E, F, S	Error - - XINVAL should not find an owner or S-state target node.	

[0045]   Referring to Table 4 and FIG. 1, when a source node (*e.g.*, source processor 12) issues a source broadcast request for data, each of the target processors or nodes (*e.g.*, target processor 14 and nodes 20) may provide a non-data response. As listed in Table 3, the cache coherency protocol employs five different types of non-data responses: a general snoop response (MISS), a snoop hit shared response (SHARED), a snoop conflict response (CONFLICT), a snoop read conflict response (RD-CONF), and a snoop hit FPP mode MAF response (FPP). It is thus possible that, in response to a source broadcast request for data, the source processor 12 can receive several different non-data responses.

The CONFLICT, RD-CONF, and FPP non-data responses help account for the fact that there may be more than one source processor issuing requests for the same data at any given time. Accordingly, the source processor 12 requester can employ a conflict state machine associated with the MAF entry for the source broadcast request to manage conflicts that may result from any given SSP broadcast request for data.

[0046]   FIG. 3 depicts an example of a conflict state diagram that represents operation of a conflict state machine that can be utilized to manage non-data responses returned to a source node. The example data state diagram of FIG. 3 implements non-data responses set forth in Table 3. As shown in the conflict state diagram of FIG. 3, an FPP response has priority over the MISS, SHARED, RD-CONF, and CONFLICT responses. Thus, the FPP response can transition the cache state machine to the FPP state, regardless of the other responses received at the source node. The CONFLICT response takes priority over the MISS, SHARED, and RD-CONF responses and thus transitions the conflict state machine to the CONFLICT state. The RD-CONF response takes priority over the MISS and SHARED responses and thus transitions the conflict state machine to the RD-CONF state. The SHARED response takes priority over the MISS response and thus transitions the conflict state machine to the SHARED state. The MISS response does not transition the state of the conflict state machine. As shown in the diagram of FIG. 3, once the conflict state machine transitions to a given state, any subsequent lower priority responses will not result in a state transition.

[0047]   In a conflict state machine (see FIG. 3) associated with a MAF, the transition to the RD-CONF state may be triggered by receiving a RD-CONF response from a snooped target node. The RD-CONF transition may also be triggered by receiving an XREADN or an XREADM request from another node. In a conflict state machine associated with a MAF at the source node, the CONFLICT transition may be triggered by receiving a CONFLICT response from a snooped node. The CONFLICT transition may also be triggered by receiving an XRDINVAL, XUPGRADE, XINVAL, or XWRITE request from another node. The CONFLICT transition may further be triggered by receiving a SET-CONF message from the data state machine associated with the MAF.

[0048]   One type of conflict situation can occur when two or more processors each have an outstanding request for the same line of data and a MAF associated with their respective requests. The response issued by a responding target processor of the group of conflicting processors depends on the MAF state for the conflicting request of the responding target processor. A list of example target processor responses that may be

14

issued in conflict cases according to the cache coherency protocol described herein is depicted below in Table 5.

TABLE 5

Source Request Type	MAF State at Target	Next MAF State at Target	Response to Source
Any Broadcast Read or Write	Any FPP Request (Except Victim)	Unchanged	FPP
	Any Victim: XINVAL XWRITE	Unchanged	CONFLICT
	Broadcast Reads: XREADN XREADM+DSM≠D-DATA* XREADC RD-CONF	Per Conflict State Machine	RD-CONF
	Broadcast Writes: XRDINVAL XUPGRADE N XREADM + DSM=D-DATA* CONFLICT	Per Conflict State Machine	CONFLICT
*DSM = Data State Machine			

[0049] As shown in Table 5, if a target node has an outstanding MAF in any FPP request state except a victim request when the source broadcast read or write request is received, the target node issues an FPP response to the source node and the target node MAF state remains unchanged. If a target node has an outstanding MAF in a FPP victim request state when the source broadcast read or write request is received, the target node issues a CONFLICT response to the source node and the target node MAF state remains unchanged. Also, if a target node has an outstanding MAF in one of the broadcast read states set forth in Table 5 when the source broadcast read or write request is received, the target node issues a RD-CONF response to the source node and the target node MAF state transitions according to the conflict state machine (see, *e.g.*, FIG. 3). Further, if a target node has an outstanding MAF in one of the broadcast write states set forth in Table 5 when the source broadcast read or write request is received, the target node issues a CONFLICT response to the source node and the target node MAF state transitions according to the conflict state machine.

[0050] After all target nodes have responded to a source broadcast read/write request issued by a source node, the action taken at the source node proceeds according to

several factors. These factors include the type of source broadcast read/write request issued by the source node, the resulting state of the data state machine (see, *e.g.*, FIG. 2), and the resulting state of the conflict state machine (see, *e.g.*, FIG. 3).

[0051] Referring back to FIG. 1, the source processor 12 may transmit a source broadcast non-migratory read snoop (XREADN, see, *e.g.*, Table 2) to the other processor 14, to the memory 16, and to the other nodes 20 via the system interconnect 18. The other nodes in the system respond to the XREADN request by providing either a data response or a non-data response (see, *e.g.*, Table 3), depending on factors such as the state of the respective nodes when the request is received and whether there is a conflict with the request, as described herein. The responses drive the data state machine and conflict state machine at the source processor associated with the XREADN request, as described herein (see, *e.g.*, FIGS. 2 and 3). Once all responses to the XREADN request have returned from the nodes in the system 10, the resulting action taken at the source processor 12 is determined in accordance with the resulting data state/conflict state combinations, such as set forth below in Table 6.

TABLE 6

Data State Machine	Conflict State Machine	Action Taken at Source Node
NO-DATA	Don't Care	Transition to FPP mode and reissue using FPP request.
Don't Care	FPP	Transition to FPP mode and reissue using FPP request.
S-DATA	NO-CONFLICT, SHARED, RD-CONF	Fill cache with S-DATA, transition cache line to S-state, and retire MAF.
S-DATA	CONFLICT	FILL-INVALID - Fill cache with S-DATA for single use, transition cache line to I-state, and retire MAF.
D-DATA	Don't Care	Error - D-DATA not returned for XREADN.
M-DATA	NO-CONFLICT, SHARED	Fill cache with M-DATA, transition cache line to E-state, F-state, or S-state, and retire MAF.
M-DATA	RD-CONF	Fill cache with M-DATA, transition cache line to S-state, and retire MAF.
M-DATA	CONFLICT	Transition to FPP mode and reissue using FPP request.

[0052] According to the cache coherency protocol described herein, an example sequence of events for an XREADN transaction is as follows:

1. Allocate an entry in a source node MAF.

2. Broadcast the XREADN commands to the home and all processors. Set the MAF entry to a SNOOPS_PENDING state.

3. Respond to snoop responses and third party snoops in accordance with the data state machine (see, *e.g.*, FIG. 2) and conflict state machine (see, *e.g.*, FIG. 3) associated with the MAF entry as well as processor snoop response Table 4.

4. After all snoop responses have returned from other nodes, take actions as determined in XREADN snoop resolution Table 6 based on the data state machine and conflict state machine associated with the MAF entry.

[0053]   The source processor 12 may also transmit a source broadcast migratory read snoop (XREADM, see, *e.g.*, Table 2) to the other processor 14, to the memory 16, and to the other nodes 20 via the system interconnect 18. The other nodes in the system respond to the XREADM request by providing either a data response or a non-data response (see, *e.g.*, Table 3), depending on factors such as the state of the respective nodes when the request is received and whether there is a conflict with the request, as described herein. The responses drive the data state machine and conflict state machine associated with the XREADM request, as described herein. After all responses to the XREADM request have returned from the nodes in the system 10, the resulting action taken at the source processor 12 is determined in accordance with the resulting data state/conflict state combinations, such as set forth below in Table 7.

TABLE 7

Data State Machine	Conflict State Machine	Action Taken at Source Node
NO-DATA	Don't Care	Transition to FPP mode and reissue using FPP request.
S-DATA	FPP	Transition to FPP mode and reissue using FPP request.
S-DATA	NO-CONFLICT, SHARED, RD-CONF	Fill cache with S-DATA, transition cache line to S-state, and retire MAF.
S-DATA	CONFLICT	FILL-INVALID - Fill cache with S-DATA for single use, transition cache line to I-state, and retire MAF.
D-DATA	NO-CONFLICT	Fill cache with D-DATA, transition cache line to D-state, and issue MACK.
D-DATA	SHARED	Fill cache with D-DATA, transition cache line to D-state, and issue MACK.
D-DATA	RD-CONF, CONFLICT	Fill cache with D-DATA, transition cache line to D-state, transition to migratory mode and issue XINVAL. Issue MACK/MACK-ACK sequence when XINVAL acknowledged.
D-DATA	FPP	Fill cache with D-DATA, transition cache line to O-state, transition to migratory mode and issue XINVAL. Issue MACK when XINVAL acknowledged. Transition to FPP and reissue using FPP request upon MACK-ACK.
M-DATA	NO-CONFLICT, SHARED	Fill cache with M-DATA, transition cache line to F-state or S-state, and retire MAF.
M-DATA	RD-CONF	Fill cache with M-DATA, transition cache line to S-state, and retire MAF.
M-DATA	CONFLICT, FPP	Transition to FPP mode and reissue using FPP request.

[0054] According to the cache coherency protocol described herein, an example sequence of events for an XREADM transaction is as follows:

1. Allocate an entry in a source node MAF.

2. Broadcast the XREADM commands to the home and all processors. Set the MAF entry to a SNOOPS_PENDING state.

3. Respond to snoop responses and third party snoops in accordance with the data state machine (see, *e.g.*, FIG. 2) and conflict state machine (see, *e.g.*, FIG. 3) associated with the MAF entry as well as processor snoop response Table 4.

4. After all snoop responses have returned from other nodes, take actions as determined in XREADM snoop resolution Table 7 based on the data state machine and conflict state machine associated with the MAF entry.

5. If XREADM snoop resolution Table 7 indicates a transition to "migratory mode," broadcast XINVAL commands to all processors.

6. Respond to third party snoops in accordance with the "Broadcast Writes" target MAF state entry of processor snoop response for conflict cases Table 5.

7. After all XINVAL responses have returned, initiate an MACK/MACK-ACK sequence.

[0055]   The source processor 12 may also transmit a source broadcast read current snoop (XREADC, see Table 2) to the other processor 14, to the memory 16, and to the other nodes 20 via the system interconnect 18. The other nodes in the system 10 respond to the XREADC request by providing either a data response or a non-data response (see Table 3), depending on factors such as the state of the respective nodes when the request is received and whether there is a conflict with the request, as described herein. The responses drive the data state machine and conflict state machine at the source processor 12 associated with the XREADC request, as described herein. After all responses to the XREADC request have returned from the nodes in the system 10, the resulting action taken at the source processor 12 is determined in accordance with the resulting data state/conflict state combinations, as set forth below in Table 8.

TABLE 8

Data State Machine	Conflict State Machine	Action Taken at Source Node
NO-DATA	Don't Care	Transition to FPP mode and reissue using FPP request.
S-DATA	FPP	Transition to FPP mode and reissue using FPP request.
S-DATA	NO-CONFLICT, SHARED, RD-CONF, CONFLICT	FILL-INVALID - Fill cache with S-DATA for single use, transition cache line to I-state, and retire MAF.
D-DATA	Don't Care	Error - D-DATA not returned for XREADC.
M-DATA	NO-CONFLICT, SHARED, RD-CONF	FILL-INVALID - Fill cache with M-DATA for single use, transition cache line to I-state, and retire MAF.
M-DATA	CONFLICT, FPP	Transition to FPP mode and reissue using FPP request.

[0056]   According to the cache coherency protocol described herein, an example sequence of events for an XREADC transaction is as follows:

1. Allocate an entry in a source node MAF.

2. Broadcast the XREADC commands to the home and all processors. Set the MAF entry to a SNOOPS_PENDING state.

3. Respond to snoop responses and third party snoops in accordance with the data state machine (see, *e.g.*, FIG. 2) and conflict state machine (see, *e.g.*, FIG. 3) associated with the MAF entry as well as processor snoop response Table 4.

4. After all snoop responses have returned from other nodes, take actions as determined in XREADC snoop resolution Table 8 based on the data state machine and conflict state machine associated with the MAF entry.

[0057]   The source processor 12 may also transmit a source broadcast read and invalidate line with owner snoop (XRDINVAL, see, *e.g.*, Table 2) to the other processor 14, to the memory 16, and to the other nodes 20 via the system interconnect 18. The other nodes in the system respond to the XRDINVAL request by providing either a data response or a non-data response (see, *e.g.*, Table 3), depending on factors such as the state of the respective nodes when the request is received and whether there is a conflict with the request, as described herein. The responses drive the data state machine and conflict state machine associated with the XRDINVAL request, as described herein. After all responses to the XRDINVAL request have returned from the nodes in the system 10, the

resulting action taken at the source processor 12 is determined in accordance with the resulting data state/conflict state combinations, as set forth below in Table 9.

TABLE 9

Data State Machine	Conflict State Machine	Action Taken at Source Node
NO-DATA	Don't Care	Transition to FPP mode and reissue using FPP request.
S-DATA	Don't Care	Error - S-DATA not returned for XRDINVAL.
Don't Care	SHARED	Error - XRDINVAL should return MISS response.
D-DATA	NO-CONFLICT, RD-CONF, CONFLICT	Fill cache with D-DATA, transition cache line to D-state, and issue MACK.
D-DATA	FPP	Fill cache with D-DATA, transition cache line to O-state, and issue MACK.
M-DATA	NO-CONFLICT, RD-CONF	Fill cache with M-DATA, transition cache line to E-state, and retire MAF.
M-DATA	CONFLICT, FPP	Transition to FPP mode and reissue using FPP request.

[0058]   According to the cache coherency protocol described herein, an example sequence of events for an XRDINVAL transaction are as follows:

1. Allocate an entry in a source node MAF.
2. Broadcast the XRDINVAL commands to the home and all processors. Set the MAF entry to a SNOOPS_PENDING state.
3. Respond to snoop responses and third party snoops in accordance with the data state machine (see, *e.g.*, FIG. 2) and conflict state machine (see, *e.g.*, FIG. 3) associated with the MAF entry as well as processor snoop response Table 4.
4. When all snoop responses have returned from other nodes, take actions as determined in XRDINVAL snoop resolution Table 9 based on the data state machine and conflict state machine associated with the MAF entry.
5. If the XRDINVAL snoop resolution Table 9 indicates an "Issue MACK" action, initiate MACK/MACK-ACK sequence.

[0059]   The source processor 12 may also transmit a source broadcast upgrade/invalidate line snoop (XUPGRADE, see, *e.g.*, Table 2) to the other processor 14, to the memory 16, and to the other nodes 20 via the system interconnect 18. The other nodes in the system respond to the XUPGRADE request by providing a non-data response

(see, *e.g.*, Table 3), depending on factors such as the state of the respective nodes when the request is received and whether there is a conflict with the request, as described herein. The responses drive the data state machine and conflict state machine associated with the XUPGRADE request, as described herein. After all responses to the XUPGRADE request have returned from the nodes in the system 10, the resulting action taken at the source processor 12 is determined in accordance with the resulting data state/conflict state combinations, such as set forth below in Table 10.

TABLE 10

Data State Machine	Conflict State Machine	Action Taken at Source Node
NO-DATA	NO-CONFLICT, RD-CONF, CONFLICT	Transition cache line to D-state, and retire MAF.
NO-DATA	SHARED	Error - XUPGRADE should return MISS response.
NO-DATA	FPP	Transition to FPP mode and reissue using FPP request.
S-DATA, D-DATA	Don't Care	Error - Data is not returned for XUPGRADE (source node is owner).
M-DATA	Don't Care	Error - No message sent to memory for XUPGRADE.

[0060]   According to the cache coherency protocol described herein, an example sequence of events for an XUPGRADE transaction is as follows:

1. Allocate an entry in a source node MAF.
2. Broadcast the XUPGRADE commands to the home and all processors. Set the MAF entry to a SNOOPS_PENDING state.
3. Respond to snoop responses and third party snoops in accordance with the data state machine (see, *e.g.*, FIG. 2) and conflict state machine (see, *e.g.*, FIG. 3) associated with the MAF entry as well as processor snoop response Table 4.
4. After all snoop responses have returned from other nodes, take actions as determined in XUPGRADE snoop resolution Table 10 based on the data state machine and conflict state machine associated with the MAF entry.

[0061]   By way of further example, assume that the processor 12 (a source node) requires a copy of data associated with a particular memory address, and assume that the data is unavailable from its own local cache 22. Since the processor 12 does not contain a copy of the requested data, the cache line of the processor may be initially in the I-state (invalid) for that data or it may contain different data altogether. For purposes of

simplicity of explanation, the starting state of the source node cache line for this and other examples is the I-state. The processor 12, operating as the source node, transmits a source broadcast non-migratory read snoop (XREADN) to the other processor 14, to the memory 16, and to the other nodes 20 via the system interconnect 18.

[0062] In this example, it is assumed that, at the time of the XREADN request, at least one other processor (*e.g.*, processor 14) in the system 10 has an outstanding XREADN request for the same data. It is further assumed that yet another processor (*e.g.*, one of the other nodes 20) is an owner node, *i.e.*, a cached ordering point for the data. For this example, assume that the owner node 20 has a copy of the data in an E-state or F-state cache line of the owner node.

[0063] Upon receiving the XREADN request broadcast from the source processor 12, the memory 16 will return an M-DATA response and the owner node 20 will return an S-DATA response (see Table 3). Upon receiving the XREADN request broadcast from the source processor 12, the target node 14 will return an RD-CONF response because the target node has a pending XREADN request for the same data (see Table 5). Referring to the data state diagram of FIG. 2, the S-DATA response from the owner node 20 has priority over the M-DATA response from memory 16. As a result, after all responses have been received from the nodes of the system 10, the data state machine associated with the XREADN request of the source processor 12 is in the S-DATA state. Referring to the conflict state diagram of FIG. 3, the RD-CONF response from the target processor 14 places the conflict state machine associated with the XREADN request of the source processor 12 in the RD-CONF state. After all responses to the XREADN request have returned from the nodes in the system 10, the resulting action taken at the source processor 12 is determined in accordance with the XREADN snoop resolution table (Table 6).

[0064] Referring to Table 6, since the data state machine is in the S-DATA state and the conflict state machine is in the RD-CONF state, the resulting action taken at the source node 12 is to fill the source node cache with the S-DATA and transition the source node cache line associated with the data to the S-state. Thus, in this example, according to the cache coherency protocol described herein, the source processor 12 cache is filled with S-DATA in response to the XREADN request, even though there is a RD-CONF with the target processor 14. The cache coherency protocol thus avoids having to transition to the FPP mode and issuance of an FPP request in this read conflict scenario because the source processor 12 cache is filled in response to the source broadcast request.

[0065] As another example, assume that the source processor 12 transmits a source broadcast non-migratory read snoop (XREADN) to the other processor 14, to the memory 16, and to the other nodes 20 *via* the system interconnect 18. In this example, it is assumed that, at the time of the XREADN request, at least one other processor (*e.g.*, processor 14) in the system 10 has an outstanding broadcast write request (*e.g.*, XRDINVAL) for the same data. It is further assumed that yet another processor (*e.g.*, one of the other nodes 20) is an owner node, *i.e.*, a cached ordering point for the data. For this example, assume that the owner node 20 has a copy of the data in an E-state or F-state cache line of the node.

[0066] Upon receiving the XREADN request broadcast from the source processor 12, the memory will return an M-DATA response and the owner node 20 will return an S-DATA response (see Table 3). Upon receiving the XREADN request broadcast from the source processor 12, the target node 14 will return a CONFLICT response because the target node has a pending XRDINVAL request for the same data (see Table 5). Referring to the data state diagram of FIG. 2, the S-DATA response from the owner node 20 has priority over the M-DATA response from memory 16. As a result, after all responses have been received from the nodes of the system 10, the data state machine associated with the XREADN request of the source processor 12 is in the S-DATA state. Referring to the conflict state diagram of FIG. 3, the CONFLICT response from the target processor 14 places the conflict state machine associated with the XREADN request of the source processor 12 in the CONFLICT state. After all responses have returned from the nodes in the system 10, the resulting action taken at the source processor 12 is determined in accordance with Table 6.

[0067] As shown in Table 6, since the data state machine is in the S-DATA state and the conflict state machine is in the CONFLICT state, the resulting action taken at the source node 12 is to FILL-INVALID, *i.e.*, fill the source node cache with the data and transition the source node cache line associated with the data to the I-state. Thus, in this example, according to the cache coherency protocol described herein, the source processor 12 cache is filled with the data, which affords the source processor a single use of the data. If the source processor 12 requires the data for further use, another SSP source broadcast read can be issued. This occurs even though there is a conflict (CONFLICT) with the target processor 14. The cache coherency protocol thus provides for avoiding transition to the FPP mode and issuance of an FPP request in this write/read conflict scenario.

24

[0068] The above examples illustrate two conflict scenarios that lead to two of the data state/conflict state combinations of Table 6. It will be appreciated that the other data state/conflict state combinations of Table 6 would similarly result in the corresponding source node actions illustrated in Table 6. It will also be appreciated that the various data state and conflict state combinations of Table 6 may arise in a virtually limitless number of circumstances involving an XREADN request with conflict and non-conflict scenarios. Regardless of the scenario under which these data state/conflict state combinations are achieved, the action taken at the XREADN source node will be determined according to the data state/conflict state combination when all responses are received at the source node. Thus, for example, if the data state machine indicates NO-DATA after all snoop responses have been received, the request is reissued in the FPP mode, as set forth in Table 6. As another example, if the conflict state machine indicates FPP (*e.g.*, another node has an outstanding FPP request for the data), the request is reissued in the FPP mode, as set forth in Table 6. As a further example, if the data state machine indicates M-DATA and the conflict state machine indicates CONFLICT, the request is reissued in the FPP mode, as set forth in Table 6.

[0069] The examples set forth above illustrate the operation of the cache coherency protocol described herein in response to an XREADN request (see Table 6). It will be appreciated that the cache coherency protocol described herein would operate in accordance with the actions set forth in Tables 7-10 in the event of a source node broadcasting an XREADM, XREADC, XRDINVAL, or XUPGRADE request, respectively. In the event that a source node broadcasts one of these requests, the target nodes of the system 10 would respond as dictated in Tables 3-5. Based on these responses, the data state machine (see, *e.g.*, FIG. 2) and conflict state machine (see, *e.g.*, FIG. 3) associated with the request would assume their respective states in the manner described herein. The action taken at the source node would be dictated by the resulting data state/conflict state combination, as set forth in the appropriate one of Tables 7-10.

[0070] FIG. 4 depicts an example of a multi-processor computing system 50. The system 50, for example, includes an SMP (symmetric multi-processor) node 52 that includes processors (P1, P2, P3, P4) 54, 56, 58 and 60 in communication with each other *via* an interconnect 62. The interconnect 62 facilitates transferring data between processors and memory of the system 50. While four processors 54, 56, 58, and 60 are depicted in the example of FIG. 2, those skilled in the art will appreciate that a greater or smaller number of processors can be implemented in the node 52.

25

[0071] Each processor 54, 56, 58, and 60 also includes an associated cache 64, 66, 68 and 70. The caches 64, 66, 68, and 70 can enable faster access to data than from an associated main memory 72 of the node 52. The system 50 implements a cache coherency protocol designed to guarantee coherency of data in the system. By way of example, the cache coherency protocol can be implemented to include a source broadcast protocol in which broadcast snoops or requests for data are transmitted directly from a source processor to all other processors and memory in the system 50. The source broadcast protocol can further be implemented in conjunction with another forward progress protocol, such as a null-directory or other directory-based protocol. The system 50 of FIG. 2, for example, employs the source broadcast protocol to process a request for data. If the request cannot be processed using the source broadcast protocol, such as where a conflict exists, the system 50 transfers to its forward progress protocol.

[0072] The memory 72 can include multiple memory modules (M1, M2, M3, M4) 74, 76, 78 and 80. For example, the memory 72 can be organized as a single address space that is shared by the processors 54, 56, 58 and 60 as well as other nodes 82 of the system 50. Each of the memory modules 74, 76, 78 and 80 can operate as a home node for predetermined lines of data stored in the memory 72. Each memory module 74, 76, 78, 80 thus can employ a table, such as a DIFT (data in flight table) (D1, D2, D3, D4) 84, 86, 88, 90, for keeping track of references that are in flight after the ordering point and for limiting the number of pending transactions to the same line allowed after the ordering point. Additionally, each of the memory modules 74, 76, 78 and 80 can include a directory (not shown), such as for use in a directory-based protocol. A coherent copy of data, for example, may reside in a home node (*e.g.*, associated with a given memory module) or, alternatively, in a cache of one of the processors 54, 56, 58 and 60.

[0073] The other node(s) 82 can include one or more other SMP nodes associated with the SMP node 52 *via* the interconnect 62. For example, the interconnect 62 can be implemented as a switch fabric or hierarchical switch programmed and/or configured to manage transferring requests and responses between the processors 54, 56, 58, and 60 and the memory 70, as well as those to and from the other nodes 82.

[0074] When a processor 56 requires desired data, the processor 56 operates as a source and issues a source broadcast snoop (*e.g.*, a broadcast read or broadcast write request) to all other processors 54, 58 and 60 as well as to memory 72 *via* the interconnect 62. The cache coherency protocol described herein is designed to ensure that a correct copy of the data is returned in response to the source broadcast snoop.

[0075] By way of example, assume that the processor 54 (a source node) requires a copy of data associated with a particular memory address, and assume that the data is unavailable from its own local cache 64. Since the processor 54 does not contain a copy of the requested data, the cache line of the processor may be initially in the I-state (invalid) for that data or it may contain different data altogether. For purposes of simplicity of explanation, the starting state of the source node cache line for this and other examples is the I-state. The processor 54, operating as the source node, transmits a source broadcast migratory read snoop (XREADM) to the other processors 56, 58, and 60, to the memory 72, and to the other nodes 82 *via* the interconnect 62.

[0076] In this example, it is assumed that, at the time of the XREADM request, at least one other processor (*e.g.*, processor 56) in the system 10 has an outstanding read request (*e.g.*, an XREADM or XREADN request) for the same data. It is further assumed that yet another processor (*e.g.*, processor 58) is an owner node, *i.e.*, a cached ordering point for the data. For this example, assume that the owner node 58 has a copy of the data in an M-state cache line.

[0077] Upon receiving the XREADM request broadcast from the source processor 12, the memory will return an M-DATA response and the owner node 58 will return a D-DATA response (see Table 3). Upon receiving the XREADM request broadcast from the source processor 54, the target node 56 may return an RD-CONF response because the target node has a pending read request for the same data (see, *e.g.*, Table 5).

[0078] Referring to the data state diagram of FIG. 2, the D-DATA response from the owner node 58 has priority over the M-DATA response from memory 72. As a result, after all responses have been received from the nodes of the system 50, the data state machine associated with the XREADM request of the source processor 54 is in the D-DATA state. Referring to the conflict state diagram of FIG. 3, the RD-CONF response from the target processor 56 places the conflict state machine associated with the XREADM request of the source processor 54 in the RD-CONF state. Once all responses to the XREADM request have returned from the nodes in the system 50, the resulting action taken at the source processor 54 is determined in accordance with the XREADM snoop resolution table (Table 7), above.

[0079] As shown in Table 7, since the data state machine is in the D-DATA state and the conflict state machine is in the RD-CONF state, the resulting action taken at the source node 54 is to fill the source node cache with the D-DATA and transition the source node cache line associated with the data to the D-state. Thereafter, the source node 54

transitions to a migratory mode, in which the node 54 broadcasts an invalidate command (XINVAL) that invalidates the cache line associated with the data at the processor 56, *i.e.*, the cache line of the processor 56 transitions to the I-state. Next, source node 54 initiates an MACK/MACK-ACK sequence to complete the ordering point migration from the owner node 58. Once the MACK-ACK response is received at the source node 54, the MAF associated with the XREADM request at the source node is retired leaving the source node cache line in the D-state. Thus, in this example, according to the cache coherency protocol described herein, the source processor 54 cache is filled with D-DATA in response to the XREADM request, even though there is a read conflict (RD-CONF) with the target processor 56. Also, in this example, the ordering point for the data migrates from the target processor 56 to the source processor 54, *i.e.*, ownership of the data transfers from the target processor 56 to the source processor 54 without updating memory. The cache coherency protocol thus provides for avoiding transition to the FPP mode and issuance of an FPP request in this read conflict scenario while providing for ordering point migration.

[0080]   The above example illustrates but a single conflict scenario that leads to one of the data state/conflict state combinations of Table 7. It will be appreciated that the other data state/conflict state combinations of Table 7 would similarly result in the corresponding source node actions illustrated in Table 7. It will also be appreciated that the various data state and conflict state combinations of Table 7 can result from a great number of XREADM circumstances involving conflict and non-conflict scenarios. The action taken at the XREADM source node will be determined according to the data state/conflict state combination after all responses have been received at the source node.

[0081]   For example, if the data state machine indicates NO-DATA after all snoop responses have been received, the request is reissued in the FPP mode, as set forth in Table 7. As another example, if the conflict state machine indicates FPP and the data state machine indicates S-DATA or M-DATA, the request is reissued in the FPP mode, as set forth in Table 7. As a further example, if the conflict state machine indicates FPP and the data state machine indicates D-DATA, the source node cache is filled with the D-DATA and transitions to the O-state. Thereafter, the source node transitions to a migratory mode, in which the node broadcasts an XINVAL that invalidates the cache line associated with the data at the other nodes. After the XINVAL is acknowledged by the other processors, an MACK/MACK-ACK sequence is initiated and, when completed, the source node transitions to the FPP mode and reissues the read request using an FPP request.

28

Alternatively, the source node could implement other forward progress techniques (*e.g.*, retrying the request in an SSP mode or employing a token based protocol).

[0082] The examples set forth above illustrate the operation of the cache coherency protocol described herein in response to an XREADM request (see, *e.g.*, Table 7). It will be appreciated that the cache coherency protocol described herein would operate in accordance with the actions set forth in Tables 6 and 8-10 in the event of a source node broadcasting an XREADN, XREADC, XRDINVAL, or XUPGRADE request, respectively. In the event that a source node broadcasts one of these requests, the target nodes of the system 50 would respond as dictated in Tables 3-5. Based on these responses, the data state machine (see, *e.g.*, FIG. 2) and conflict state machine (see, *e.g.*, FIG. 3) associated with the request would assume their respective states in the manner described herein. The action taken at the source node would be dictated by the resulting data state/conflict state combination, as set forth in the appropriate one of Tables 6 and 8-10.

[0083] FIG. 5 depicts an example of another multi-processor system 100 that includes a plurality of processors 102, 104 and 106 in communication with each other *via* a switch fabric 108. The system 100 also includes associated memory 110, which can be organized as a single address space that is shared by the processors 102, 104, and 106. For example, the memory 110 can be implemented as a plurality of separate memory modules associated with each of the respective processors 102, 104, and 106 for storing data. The system 100, for example, can be implemented as an integrated circuit or as circuitry containing plural integrated circuits.

[0084] The system 100 can employ a source broadcast or source-snoopy cache coherency protocol. For this type of protocol, a source processor 102, 104, and 106 can issue a source broadcast request to all other processors in the system and to the memory 110. In the event that conflict arises, or the source broadcast request otherwise fails, the source processor can employ a forward progress technique to complete the transaction. For example, the source processor can transfer to a forward-progress protocol, such as a null-directory or other directory-based protocol, and reissue the request using such protocol.

[0085] In a null-directory-based protocol, for example, the memory 110 includes home nodes for each cache line. Instead of issuing a broadcast to all cache targets, the source issues a single request to the home node for such data. The home node thus operates as static ordering point for requested data since all requests are sent to the home

node for ordering before snoops are broadcast. This tends to add an additional hop for the majority of references compared with a broadcast-based protocol described above. If the system employs a standard directory-based protocol, ordering is implemented, but the memory 110 employs associated directories that facilitate locating the data (*e.g.*, based on the directory state associated with the requested data). In a standard directory protocol, there will be times when the directory can indicate that there are no cached copies, and thus the home node can respond with the data without issuing any snoops to the system 100.

[0086]   The processor 102 includes cache memory 114 that contains a plurality of cache lines 116 (*e.g.*, lines 1-M, where M is a positive integer, M ≥ 1). Each cache line 116 can contain one or more memory blocks. A tag address (ADDRESS) is associated with the data contained in each cache line 116. Additionally, each cache line 116 can contain state information identifying the state of the data contained at that cache line. Examples of states that can be associated with each cache line 116 are identified above in Table 1.

[0087]   A cache controller 118 is associated with the cache memory 114. The cache controller 118 controls and manages access to the cache memory, including requests for data and responses. The cache controller 118 communicates requests and responses *via* a switch interface 120 that is coupled with the switch fabric 108. The switch interface 120, for example, includes an arrangement of queues (*e.g.*, input and output queues) or other data structures that organize both requests and responses issued by the processor 102 as well as requests and responses for execution by the processor.

[0088]   In the example of FIG. 5, the cache controller 118 includes a state engine 122 that controls the state of each respective line 116 in the cache memory 114. The state engine 122 is programmed and/or configured to implement state transitions for the cache lines 116 based on predefined rules established by the cache coherency protocol described herein. For example, the state engine 122 can modify the state of a given cache line 116 based on requests issued by the processor 102. Additionally, the state engine 122 can modify the state of a given cache line 116 based on responses received at the processor 102 for the given tag address, such as may be provided by another processor 104, 106 and/or the memory 110.

[0089]   The cache controller 118 also includes a request engine 124 that sends requests to the system 100. The request engine 124 employs a miss address file (MAF)

30

126 that contains MAF entries for outstanding requests associated with some subset of the locations in the cache memory 114. The MAF can be implemented as a table, an array, a linked list or other data structure programmed to manage and track requests for each cache line. For example, when the processor 102 requires data associated with a given tag address for a given line 116, the request engine 124 creates a corresponding entry in the MAF 126. The MAF entry includes fields that identify, for example, the tag address of the data being requested, the type of request, and response information received from other nodes in response to the request. The request engine 124 thus employs the MAF 126 to manage requests issued by the processor 102 as well as responses to such requests. The request engine can employ a data state machine and conflict state machine (see, *e.g.*, FIGS. 2 and 3) associated with each MAF entry for helping to manage a data state and a conflict state associated with each MAF entry.

[0090] The cache controller 118 also includes a response engine 128 that controls responses provided by the processor 102. The processor 102 provides responses to requests or snoops received *via* the switch interface 120 from another processor 104 and 106 or memory 110. The response engine 128, upon receiving a request from the system 100, cooperates with the state engine 122 and the MAF 126 to provide a corresponding response based on the type of request and the state of data contained in the cache memory 114. For example, if a MAF entry exists for a tag address identified in a request received from another processor or memory, the cache controller can implement appropriate conflict resolution defined by the coherency protocol. The response engine thus enables the cache controller to send an appropriate response to requesters in the system 100. A response to a request can also cause the state engine 122 to effect a state transition for an associated cache line 116.

[0091] By way of example, assume that the processor 102 requires data not contained locally in its cache memory 114. The request engine 124 will create a MAF entry in the MAF 126, corresponding to the type of request and the tag address associated with data required. In this example, assume that the processor 102 issues a broadcast read and invalidate line request (XRDINVAL, see Table 2) and a corresponding entry in the MAF 126. Assume also that the processor 104 is an owner node for the data and includes the data in a D-state cache line. Assume further that the processor 106 has an outstanding XRDINVAL MAF for the same data. The cache controller 118 broadcasts a source snoop XRDINVAL request to the nodes of the system 100 *via* the switch interface 120 and switch fabric 108.

[0092] In response to receiving the XRDINVAL request from the source node 102, the memory 110 provides an M-DATA response. The owner node 104 provides a D-DATA response and transitions to the T-state in accordance with the data migration procedures of the cache coherency protocol (see Table 4). The processor 106, having an outstanding XRDINVAL MAF for the data, responds to the XRDINVAL by providing a non-data CONFLICT response (see Table 5).

[0093] Referring to the data state diagram of FIG. 2, the D-DATA response from the owner node 104 has priority over the M-DATA response from memory 110. As a result, after all responses have been received from the nodes of the system 100, the data state machine associated with the MAF entry for the XRDINVAL request of the source node 102 is in the D-DATA state. Referring to the conflict state diagram of FIG. 3, the CONFLICT response from the processor 106 causes the conflict state machine associated with the XRDINVAL request of the source processor 102 to transition to the CONFLICT state. After all responses to the XRDINVAL request have returned from the nodes in the system 100, the resulting action taken at the source processor 102 can be determined in accordance with the XRDINVAL snoop resolution Table 9.

[0094] As shown in Table 9, since the data state machine is in the D-DATA state and the conflict state machine is in the CONFLICT state, the resulting action taken at the source node 102 is to fill the source node cache with the D-DATA and transition the source node cache line associated with the data to the D-state. Thereafter, the source node 102 issues an MACK to the node 104. Upon receiving an MACK-ACK response from the node 104, the node 104 retires the MAF and thus becomes the owner node for the data. Thus, in this example, according to the cache coherency protocol described herein, the source processor 102 cache is filled with D-DATA in response to the XRDINVAL request, even though there is a conflict (CONFLICT) with the processor 106. The cache coherency protocol thus provides for avoiding transition to the FPP mode and issuance of an FPP request in this read conflict scenario.

[0095] The above example illustrates but a single conflict scenario that leads to one of the data state/conflict state combinations of Table 9. It will be appreciated that the other data state/conflict state combinations of Table 9 can result in the corresponding source node actions illustrated in Table 9. It will also be appreciated that the various data state and conflict state combinations of Table 9 can result from a great number of XRDINVAL circumstances involving conflict and non-conflict scenarios. Regardless of the scenario under which these data state/conflict state combinations are achieved, the

32

action taken at the XRDINVAL source node will be determined according to the data state/conflict state combination after all responses are received at the source node.

[0096] For example, if the data state machine indicates NO-DATA after all snoop responses have been received, the request is reissued in the FPP mode, as set forth in Table 9. As another example, if the conflict state machine indicates FPP and the data state machine indicates M-DATA, the request is reissued in the FPP mode, as set forth in Table 9. As a further example, if the conflict state machine indicates FPP and the data state machine indicates D-DATA, the source node cache is filled with the D-DATA and transitions to the O-state. Thereafter, the source node initiates an MACK/MACK-ACK sequence and, when completed, the source node transitions to the FPP mode and reissues the write request using an FPP request.

[0097] The examples set forth above illustrate the operation of the cache coherency protocol described herein in response to an XRDINVAL request (see Table 9). It will be appreciated that the cache coherency protocol described herein would operate in accordance with the actions set forth in Tables 6-8, and 10 in the event of a source node broadcasting an XREADN, XREADM, XREADC, or XUPGRADE request, respectively. In the event that a source node broadcasts one of these requests, the target nodes of the system 100 would respond as dictated in Tables 3-5. Based on these responses, the data state machine (see FIG. 2) and conflict state machine (see FIG. 3) associated with the request would assume their respective states in the manner described herein. The action taken at the source node would be dictated by the resulting data state/conflict state combination, as set forth in the appropriate one of Tables 6-8 and 10.

[0098] The various examples of conflict scenarios depicted herein so far have been addressed from the perspective of only one of the conflicting processors in a given conflict scenario and considering the conditions at the other node to be essentially static. These examples have not addressed the fact that in a conflict scenario, the source node and target node designations are relative. To illustrate this point, consider two processors, A and B, each of which have outstanding requests for the same data and therefore conflict with each other. From the point of view of processor A, processor A is the source node and processor B is the target node. From the point of view of processor B, processor B is the source node and processor A is the target node. It will thus be appreciated that in conflict scenarios, conflicting requests are handled by the cache coherency protocol at both conflicting nodes in the manner described herein. It will also be appreciated that the manner in which the requests of the conflicting processors are handled can depend in large

part on the timing of the creation and/or retirement of the respective MAF entries at the conflicting processors and the timing of the respective snoops/responses of the conflicting processors.

[0099] In view of the foregoing structural and functional features described above, certain methods that can be implemented using a coherency protocol will be better appreciated with reference to FIGS. 6-11. FIGS. 6-10 depict various example timing diagrams for conflict scenarios that can arise in a multi-processor system employing a cache coherency protocol as described herein. Each of the examples illustrates various interrelationships between requests and responses and state transitions that can occur for a given memory tag address in different memory devices or caches. In each of these examples, time flows in the direction of an arrow labeled "TIME." Those skilled in the art may appreciate various other conflict scenarios that can arise in a multi-processor system employing a cache coherency protocol as described herein.

[00100] FIG. 6 illustrates a network 160 that includes processor nodes 162, 164, and 166 and a home node 168. Initially, nodes 162 and 164 are in an I-state for a particular cache line and node 166 is in the E-state for the cache line. The home node 168 contains a memory copy of the data associated with the cache line. In this example case, node 162 allocates a read MAF entry (RDMAF) 170 and, thereafter, node 164 allocates a read MAF entry 172 for the requested data. Next, node 164 receives a read conflict (RD-CONF) response (see, e.g., Table 5) to a non-migratory read request (XREADN) broadcast from node 164 to node 162. Next, node 162 receives an S-DATA response to an XREADN request broadcast from node 162 to node 166. Node 166, upon providing the S-DATA response to node 162, transitions to the first among equals state (F-state, see, e.g., Table 4). Next, node 162 receives an M-DATA response to an XREADN request broadcast from node 162 to home node 168. Next, node 164 receives an M-DATA response to an XREADN request broadcast from node 164 to home node 168. Thereafter, node 166 silently evicts the cache line and transitions the cache line to invalid (I-state), as indicated at 174. Next, node 164 receives a MISS response to XREADN request broadcast from node 164 to node 166 (because the XREADN request broadcast by node 164 found the I-state cache line at node 166).

[00101] At this point, responses have been received from all of the nodes to which node 164 broadcast snoop requests. Referring to FIG. 2, the data state machine for the MAF 172 at node 164, having received the M-DATA response from the home node 168 and no other data responses, transitions to the M-DATA state. Referring to FIG. 3, the

34

conflict state machine for the MAF 172 at node 164, having received the RD-CONF response from node 162 and the MISS response from node 166, transitions to the RD-CONF state. Referring to the XREADN snoop resolution table (Table 6), for the data state/conflict state combination of M-DATA and RD-CONF, the action taken at the source node 164 for the XREADN MAF 172 is to fill the cache line with the M-DATA, transition the cache line to the shared state (S-state), and retire the MAF 172. Thus, according to the cache coherency protocol described herein, in this read conflict scenario, the cache line of node 164 is filled with a shared copy of the requested data, even though there is a conflicting read request from another node.

[00102]   After node 164 has transitioned to the S-state, node 162 receives a SHARED response to an XREADN request broadcast from node 162 to node 164. At this point, responses have been received from all of the nodes to which node 162 broadcast snoop requests. Referring to FIG. 2, the data state machine for the MAF 170 at node 162, having received the M-DATA response from the home node 168 and S-DATA from node 166, transitions to the S-DATA state. Referring to FIG. 3, the conflict state machine for the MAF 170 at node 162, having received the SHARED response and the XREADN request from node 164, transitions to the RD-CONF state due to the XREADN request. Referring to Table 6, for the data state/conflict state combination of S-DATA and RD-CONF, the action taken at the source node 162 for the XREADN MAF 170 is to fill the cache line with the S-DATA, transition the cache line to the S-state, and retire the MAF 170. Thus, according to the cache coherency protocol described herein, in this read conflict scenario, the cache line of node 162 is filled with a shared copy of the requested data, even though there is a conflicting read request from another node.

[00103]   FIG. 7 illustrates an example scenario in which a network 180 includes processor nodes 182, 184, and 186 and a home node 188. Initially, nodes 182, 184, and 186 are in an I-state for a particular cache line and the home node 188 contains a memory copy of the data associated with the cache line. In this example case, node 184 allocates a read MAF entry 192 and broadcasts an XREADN request to node 182, which returns a MISS response indicating that the cache line is invalid at node 182. Next, node 182 allocates a read MAF entry 190 and broadcasts an XREADN request to node 184, which returns a RD-CONF response. Next, node 182 receives a MISS response to an XREADN request broadcast from node 182 to node 186, indicating that the cache line for the requested data at node 186 is invalid. Next, node 182 receives an M-DATA response to an XREADN request broadcast from node 182 to home node 188.

[00104] At this point, responses have been received from all of the nodes to which node 182 broadcast snoop requests. Referring to FIG. 2, the data state machine for the MAF 190 at node 182, having received the M-DATA response from the home node 188 and no other data responses, transitions to the M-DATA state. Referring to FIG. 3, the conflict state machine for the MAF 190 at node 182, having received the RD-CONF response from node 184 and the MISS response from node 186, transitions to the RD-CONF state. Referring to Table 6, for the data state/conflict state combination of M-DATA and RD-CONF, the action taken at the source node 182 for the XREADN MAF 190 is to fill the cache line with the M-DATA, transition the cache line to the S-state, and retire the MAF 192. Thus, according to the cache coherency protocol described herein, in this read conflict scenario, the cache line of node 182 is filled with a shared copy of the requested data, even though there is a conflicting read request from another node.

[00105] After node 182 has transitioned to the S-state, node 184 receives an M-DATA response to an XREADN request broadcast from node 184 to home node 188. Next, node 184 receives a MISS response to an XREADN request broadcast from node 184 to node 186. At this point, responses have been received from all of the nodes to which node 184 broadcast snoop requests. Referring to FIG. 2, the data state machine for the MAF 192 at node 184, having received the M-DATA response from the home node 188 and no other data responses, transitions to the M-DATA state. Referring to FIG. 3, the conflict state machine for the MAF 192 at node 184, having received the MISS responses from node 182 and node 186 and the XREADM request from node 190, transitions to the RD-CONF state. Referring to Table 6, for the data state/conflict state combination of M-DATA and RD-CONF, the action taken at the source node 184 for the XREADN MAF 192 is to fill the cache line with the M-DATA, transition the cache line to the S-state, and retire the MAF 192. Thus, according to the cache coherency protocol described herein, in this read-conflict scenario, the cache line of node 182 is filled with a shared copy of the requested data, even though there is a conflicting read request from another node. It should be noted that, in the example illustrated in FIG. 7, even though there were conflicting MAFs 190 and 192, node 184 never "saw" the conflict (*i.e.*, never received a RD-CONF response from node 180) because of the timing of the events illustrated in FIG. 7.

[00106] FIG. 8 illustrates an example scenario in which a network 200 includes processor nodes 202, 204, and 206 and a home node 208. Initially, nodes 202 and 204 are in an I-state for a particular cache line and node 206 is in the S-state for the cache line.

The home node 208 contains a memory copy of the data associated with the cache line. In this example case, node 204 allocates a write MAF entry (WRMAF) 212 and broadcasts a read and invalidate line with owner request (XRDINVAL) to node 202, which returns a MISS response indicating that the cache line is invalid at node 202. Next, node 202 allocates a write MAF entry 210 and broadcasts an XRDINVAL request to node 204, which returns a CONFLICT response indicating that there is a write conflict with the outstanding MAF 212 at node 204. Next, node 202 receives a MISS response to an XRDINVAL request broadcast from node 202 to node 206. The cache line for the requested data at node 206 transitions to the I-state. Next, node 202 receives an M-DATA response to an XRDINVAL request broadcast from node 202 to home node 208.

[00107] At this point, responses have been received from all of the nodes to which node 202 broadcast snoop requests. Referring to FIG. 2, the data state machine for the MAF 210 at node 202, having received the M-DATA response from the home node 208 and no other data responses, transitions to the M-DATA state. Referring to FIG. 3, the conflict state machine for the MAF 210 at node 202, having received the CONFLICT response from node 204 and the MISS response from node 206, transitions to the CONFLICT state. Referring to the XRDINVAL snoop resolution table (Table 9), for the data state/conflict state combination of M-DATA and CONFLICT, the action taken at the source node 202 for the XRDINVAL MAF 210 is to transition to the FPP mode and reissue the request using an FPP request, as indicated at 214. Thus, in this write conflict scenario shown in the example of FIG. 8, the cache coherency protocol described herein forces node 202 to transition to the FPP mode due to the conflicting write request from node 204.

[00108] After node 206 has transitioned to the I-state, node 204 receives an M-DATA response to an XRDINVAL request broadcast from node 204 to home node 208. Next, node 204 receives a MISS response to an XRDINVAL request broadcast from node 204 to node 206, node 206 having already been invalidated by the XRDINVAL request from node 202. At this point, responses have been received from all of the nodes to which node 204 broadcast snoop requests. Referring to FIG. 2, the data state machine for the MAF 212 at node 204, having received the M-DATA response from the home node 208, transitions to the M-DATA state. Referring to FIG. 3, the conflict state machine for the MAF 212 at node 204, having received the MISS response from nodes 202 and 206 and the XRDINVAL request from node 210, transitions to the CONFLICT state due to the XRDINVAL request. Referring to Table 9, for the data state/conflict state combination of

37

M-DATA and CONFLICT, the action taken at the source node 204 for the XRDINVAL MAF 212 is to transition to the FPP mode and reissue the request using an FPP request, as indicated at 216. Thus, according to the cache coherency protocol described herein, in this write conflict scenario, the cache coherency protocol described herein forces node 204 to transition to the FPP mode due to the conflicting write request from node 202. It should be noted that, in the example illustrated in FIG. 8, even though there were conflicting MAFs 210 and 212, node 204 never "saw" the conflict (*i.e.*, never received a CONFLICT response from node 200) because of the timing of the communications between the nodes.

[00109] FIG. 9 illustrates an example scenario in which a network 220 includes processor nodes 222, 224, and 226 and a home node 228. Initially, nodes 222, 224, and 226 are in an I-state for a particular cache line and home node 228 contains a memory copy of the data associated with the cache line. In this example case, node 222 allocates a WRMAF entry 230 and, thereafter, node 224 allocates a read MAF entry 232. Node 224 receives a CONFLICT response to an XREADN request broadcast from node 224 to node 222, due to the pending write MAF 230 at node 222. Next, node 222 allocates a write MAF entry 230 and broadcasts an XRDINVAL request to node 226, which returns a MISS response acknowledging the invalidate line request. Next, node 222 receives an M-DATA response to an XRDINVAL request broadcast from node 222 to home node 228. Next, node 222 receives a RD-CONF response to an XRDINVAL request broadcast from node 222 to node 224, due to the pending read MAF 232 at node 224.

[00110] At this point, responses have been received from all of the nodes to which node 222 broadcast snoop requests. Referring to FIG. 2, the data state machine for the MAF 230 at node 222, having received the M-DATA response from the home node 228 and no other data responses, transitions to the M-DATA state. Referring to FIG. 3, the conflict state machine for the MAF 230 at node 222, having received the RD-CONF response from node 224 and the MISS response from node 226, transitions to the RD-CONF state. Referring to Table 9, for the data state/conflict state combination of M-DATA and RD-CONF, the action taken at the source node 222 for the XRDINVAL MAF 230 is to fill the cache line with the M-DATA, transition the cache line to the E-state, and retire the MAF 230. Thus, according to the cache coherency protocol described herein, in this write/read conflict scenario, the cache line of node 222 (the writer) is filled with an exclusive copy of the requested data, even though there is a conflicting read request from another node.

[00111]   Meanwhile, node 224 receives an M-DATA response to an XREADN request broadcast from node 224 to home node 228. Next, node 224 receives a MISS response to an XREADN request broadcast from node 224 to node 226. At this point, responses have been received from all of the nodes to which node 224 broadcast snoop requests. Referring to FIG. 2, the data state machine for the MAF 232 at node 224, having received the M-DATA response from the home node 228, transitions to the M-DATA state. Referring to FIG. 3, the conflict state machine for the MAF 232 at node 224, having received the MISS response from node 226 and the CONFLICT response from node 222, transitions to the CONFLICT state. Referring to Table 6, for the data state/conflict state combination of M-DATA and CONFLICT, the action taken at the source node 224 for the XREADN MAF 232 is to transition to the FPP mode and reissue the request using an FPP request, as indicated at 234. Thus, in the write/read conflict scenario of the example of FIG. 9, the cache coherency protocol described herein forces node 224 (the reader) to transition to the FPP mode due to the conflicting write request from node 222.

[00112]   FIG. 10 illustrates an example scenario in which a network 240 includes processor nodes 242, 244, and 246 and a home node 248. Initially, node 242 is in an owner state (O-state) for a particular cache line, node 244 is in an I-state for the cache line, and node 246 is in an S-state for the cache line. Home node 248 contains a memory copy of the data. In this example case, node 242 allocates a WRMAF entry 250 and, thereafter, node 244 allocates a RDMAF entry 252. Next, node 242 broadcasts an upgrade/invalidate line request (XUPGRADE) request to node 244, which, having a pending read MAF 252, returns a RD-CONF response. Next, node 242 receives a MISS response to an XUPGRADE request broadcast from node 242 to node 246. Node 246 transitions to the I-state in response to the XUPGRADE request from node 242. Next, node 244 receives a CONFLICT response to an XREADN request broadcast from node 244 to node 242. Node 242 returns the CONFLICT response because node 242 has a pending XUPGRADE WRMAF 250.

[00113]   At this point, responses have been received from all of the nodes to which node 242 broadcast snoop requests. Note that, by definition, an XUPGRADE snoop is not broadcast to home node 248. Thus, responses to all snoops have been received at source node 242. Referring to FIG. 2, the data state machine for the MAF 250 at node 242, having received no data responses, remains in the NO_DATA state. Referring to FIG. 3, the conflict state machine for the MAF 250 at node 242, having received the RD-CONF response from node 244 and the MISS response from node 246, transitions to the RD-

CONF state. Referring to the XUPGRADE snoop resolution table (Table 10), for the data state/conflict state combination of NO_DATA and RD-CONF, the action taken at the source node 242 for the XUPGRADE MAF 250 is to is to transition the cache line to the D-state, and retire the MAF 250. Thus, according to the cache coherency protocol described herein, in this write/read conflict scenario, node 242 (the writer) is permitted to invalidate the cache line, even though there is a conflicting read request from another node.

[00114] Meanwhile, node 244 receives an M-DATA response to an XREADN request broadcast from node 244 to home node 248. Next, node 244 receives a MISS response to an XREADN request broadcast from node 244 to node 246, node 246 having transitioned to the I-state. At this point, responses have been received from all of the nodes to which node 244 broadcast snoop requests. Referring to FIG. 2, the data state machine for the MAF 252 at node 244, having received the M-DATA response from the home node 248, transitions to the M-DATA state. Referring to FIG. 3, the conflict state machine for the MAF 252 at node 244, having received the MISS response from node 246 and the CONFLICT response from node 242, transitions to the CONFLICT state. Referring to Table 6, for the data state/conflict state combination of M-DATA and CONFLICT, the action taken at the source node 244 for the XREADN MAF 252 is to transition to the FPP mode and reissue the request using an FPP request, as indicated at 254. Thus, in the write/read conflict scenario shown in the example of FIG. 10, the cache coherency protocol described herein forces node 244 (the reader) to transition to the FPP mode due to the conflicting write request from node 242.

[00115] FIG. 11 depicts a method that includes providing a source broadcast request from a first node for data, as indicated at 300. The method also includes providing a read conflict response to the first node from a second node in response to the source broadcast request from the first node, as indicated at 310. The read conflict response shown at 310 indicates that the second node has a pending broadcast read request for the data. The method also includes providing the requested data to the first node from a third node in response to the source broadcast request from the first node, as indicated at 320. The method further includes placing the data provided by the third node in a cache associated with the first node, as indicated at 330.

[00116] What have been described above are examples of the present invention. It is, of course, not possible to describe every conceivable combination of components or methodologies for purposes of describing the present invention, but one of ordinary skill

in the art will recognize that many further combinations and permutations of the present invention are possible. Accordingly, the present invention is intended to embrace all such alterations, modifications and variations that fall within the spirit and scope of the appended claims.